Experiences Porting Mini-applications to OpenACC and OpenMP on Heterogeneous Systems

Verónica G. Vergara Larrea*, Reuben D. Budiardja*, Rahulkumar Gayatri†,
Christopher Daley†, Oscar Hernandez* and Wayne Joubert*
*National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN
Email: {vergaravg,reubendb,oscar,joubert}@ornl.gov

†National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Berkeley, CA
Email: {rgayatri,csdaley}@lbl.gov

***Abstract*—This paper studies mini-applications–minisweep, GenASIS, GPP, and FF–that use computational methods commonly encountered in HPC. We will port these applications to develop OpenACC and OpenMP versions, and evaluate their performance on Titan (Cray XK7/K20x GPUs), Cori (Cray XC40/Intel KNL), Summit (IBM Power9/Volta GPUs), and Cori-GPU (Cray CS-Storm 500NX/Intel Skylake and Volta GPUs). Our goals are for these new ports to be useful to both application and compiler developers, to document and describe the lessons learned and the methodology to create optimized OpenMP and OpenACC versions, and to provide a description of possible migration paths between the two specifications. Cases where specific directives or code patterns result in improved performance for a given architecture will be highlighted. We also include discussions of the functionality and maturity of the latest compilers available on the above platforms with respect to OpenACC or OpenMP implementations.**

***Keywords*-OpenMP; OpenACC; performance evaluation; mini-applications;**

I. Introduction

For the last few years, two distinct architectural trends have remained the focus of exascale efforts: one relying on many-core CPU and another relying on attached GPU accelerators. The November 2018 edition from the Top500 list [1] shows that that GPU accelerated computing has gained a significant portion of the market. Five out of the top ten systems in the list are using GPU accelerators including the top two systems, Summit at Oak Ridge National Laboratory (ORNL) and Sierra at Lawrence Livermore National Laboratory (LLNL). More recently, the National Energy Research Scientific Computing Center (NERSC) announced that their next generation system, Perlmutter—expected to have approximately 3X the performance of Cori—will be a hybrid system that will include a partition with GPU accelerators. Given the increasing complexity of future architectures, it is important for application developers to keep performance portability in mind when choosing a programming model.

OpenACC and OpenMP are two of the most commonly used directives-based APIs for in-node parallelization. Offload support for accelerators was first introduced by OpenACC which has provided many application developers a preview for directive-based programming for accelerators and multi-cores with relative ease. OpenMP followed by introducing an accelerator programming model in the OpenMP 4.0 specification published in 2014 as part of a bigger specification providing many different types of parallelism (e.g., tasks, offload, worksharing, SIMD, etc). A growing list of compilers are implementing support for OpenMP offloading to attached accelerators. Currently, many of the mostpopular compilers—including the Intel, GCC, Clang, CCE and XL compilers—used by high performance computing (HPC) centers provide some level of support for the OpenMP accelerator model.

The two APIs differ in their approach to specify parallelization. OpenACC is generally considered to be more descriptive, meaning that programmers can rely on the compiler implementation to determine the best way to parallelize the code for a particular target. OpenMP 4.5, on the other hand, is considered to be a prescriptive programming model and requires the programmer to explicitly specify how code

should be parallelized. Although a prescriptive approach gives the programmer more control, it can be less portable from a performance perspective as different architectures will require different clauses. To address this gap, OpenMP 5.0 introduced the loop directive to provide an alternative that would allow the compiler to choose the optimizations to use.

In an effort to better understand the impact to application performance and to evaluate the level of support provided by implementations between these two programming models, researchers have conducted various studies comparing OpenACC vs. OpenMP versions of several kernels including Jacobi [2] and common HPC kernels such as those available in the SPEC/HPG benchmark suite [3], [4]. Other papers have explored the use of OpenACC on mini-applications to explore performance portability [5]. In [6] the authors show that through the use of directives it is possible to get comparable performance to optimized hand-written CUDA codes with some code restructuring. Directives have also been used successfully to port real applications including a hybird version of S3D, a combustion modeling application, that showed a decent speedup using OpenACC [7].

Building on that work, for this study, we have selected four (mini-)applications—minisweep, GENASIS GPP, and FF—that use computational methods commonly encountered in HPC. They also provide a partial representation of workloads executed at the Oak Ridge Leadership Computing Facility (OLCF) and NERSC. Minisweep [8] is a proxy application for Denovo [9], a radiation transport code. Minisweep is written in C and includes support for CUDA, and more recently support for OpenACC [10]. GENASIS is a multi-physics simulation code with large-scale astrophysics simulations as its primary target applications [11]. It is written using features of modern Fortran standards (Fortran 2003 and 2008). Its modular design allows mini-applications to be built using only a portion of the code [12], which has recently been ported to the OpenMP accelerator programming model [13]. GPP and FF are mini-apps extracted from the BerkeleyGW material science code [**?**]. BerkeleyGW is written predominantly in Fortran-90 and computes electron excited-state properties using the ab initio GW approximation [14]. The GPP and FF mini-apps represent the General Plasmon Pole and Full Frequency self-energy summations in BerkeleyGW. The GPP mini-app already has documented OpenMP 4.5 and OpenACC ports.

For this project, we port the aforementioned applications so that each has both OpenACC and OpenMP versions. We evaluate application performance of both the OpenMP and OpenACC versions on distinct platforms including Titan (Cray XK7/K20x GPUs), Cori (Cray XC40/Intel KNL), Summit (IBM Power9/Volta GPUs), Summitdev (IBM Power8/Pascal GPUs), and Cori-GPU (Cray CS-Storm 500NX/Intel Skylake and Volta GPUs) using different compilers. Our objectives are threefold: to contribute new ports of key mini-applications that can be useful both as examples for other application developers and to compiler developers, to document and describe the lessons learned and the methodology to create optimized OpenMP and OpenACC versions, and to provide a description of possible migration paths between the two specifications. As part of this work, we highlight cases where specific directives or code patterns result in improved performance for a given architecture. Finally, by way of this experience, we document and discuss functionality and maturity of the latest compilers available on Titan, Cori, Summit, and Cori-GPU with respect to their OpenACC or OpenMP implementations. We make available codes resulting from this work in an open repository.

The rest of this paper is organized as follows. In Section II we describe the porting efforts for each application for both OpenACC and OpenMP directive and demonstrate the results. Section III collects the leason learned from this work and ends with concluding remarks.

II. APPLICATIONS: PORTING AND RESULTS

For this work, we study the performance of each individual mini-application–Minisweep, GenASIS, GPP, and FF–on Titan, Summitdev, Summit, Cori, Cori-GPU.

Titan is a Cray XK7 system with 18,688 compute nodes each with a 16-core AMD Opteron processor and an NVIDIA K20x GPU. Summitdev is a development system available at the OLCF which is comprised of 54 nodes each with two 10-core IBM POWER8 processors and four NVIDIA P100 GPUs. Summit is the OLCF's latest flagship supercomputer and number one system in the Top500 [15] November 2018 list. Summit is an IBM system with 4,608 compute nodes each with two 22-core IBM POWER9 processors and six NVIDIA V100 GPUs. Cori is a Cray XC40 system with both Intel multi-core and many-core nodes. It has 2,388 multi-core nodes which consist of dual-socket Intel Xeon E5-2698 v3 'Haswell' CPUs with 16 cores per socket running at 2.3 GHz and 9,688 many-core nodes which consist of a single 68-core Intel Xeon-Phi 7250 'KNL' CPU running at 1.4 GHz. The nodes in the system are connected with a Cray Aries interconnect. Cori-GPU is an 18-node Cray CS-Storm 500NX system. Each node consists of a dual-socket Intel Xeon Gold 6148 'Skylake' CPU and 8 NVIDIA Volta V100 GPUs. Each Xeon socket has 20 cores running at a clock frequency of 2.4 GHz. The V100 GPUs are connected to each other via NVLink, the CPUs and GPUs are connected via a PCIe switch, and the nodes are connected in a fat-tree configuration with QDR Infiniband.

A. Minisweep

The Minisweep mini-appication [8], part of the Profugus radiation transport proxy application project [16], models the computational pattern of the sweep kernel used in the Denovo S_n radiation transport application [9]. Denovo solves the six-dimensional linear steady-state Boltzmann

transport equation, with applications to nuclear reactor core analysis (neutronics), nuclear forensics, radiation shielding and radiation detection. The S_n sweep kernel of Denovo accounts for 80-99% of the runtime of the Denovo code, thus is a significant focus of computational acceleration efforts.

The Minisweep algorithm employs multidimesional parallelism and is particularly challenging computationally due to its recursive wavefront computation. It contains multiple computational moifs, posing a challenge to directives-based programming models, including dense and sparse linear algebra, structured grids, halo communications, hierarchical synchronizations and atomic updates.

Minisweep is written in C and supports CUDA and OpenMP 3.1. Recently, Searles et al. [10] ported Minisweep to OpenACC, with favorable performance results on a par with the CUDA implementation. For this work, we ported Minisweep to thew OpenMP 4 accelerator programming model.

Minisweep's algorithm utilizes a 3-D grid of n_x, n_y, n_z dimensions. The code supports different values for the number of angular directions (n_a), energy groups (n_e), unknowns per gridcell (n_u), and a compile-time variable `NM` for the number of moments.

The algorithm is structured as a nest of multiple loop levels. At the top level is a loop over the eight octant directions; these are independent, parallelizable loop iterations but require an atomic update in the innermost loop to avoid a race condition. The octant loop contains three more nested loops: a sequential wavefront loop and a pair of loops over x and y coordinates in the wavefront. At the innermost level are three code blocks, each a loop nest over several remaining problem dimensions.

1) Porting Minisweep to OpenMP 4.5: For this work, we ported Minisweep to support the OpenMP 4.5 accelerator programming model using the OpenACC version developed in [10] as a base.

```
#pragma omp target teams distribute parallel for collapse(3)
for( ie=0; ie<dim_ne; ++ie )
  for( iu=0; iu<NU; ++iu )
    for( ia=0; ia<dim_na; ++ia )
      for( im=0; im < dim_nm; ++im )
      {
        result += ...;
        vs_local[<ind>] = result;
      }
#pragma omp target update from(vs_local[0:vs_local_size])

for( ie=0; ie<dim_ne; ++ie )
  for( ia=0; ia<dim_na; ++ia )
  {
    Quantities_solve_inline(vs_local, dims, facexy, facexz, faceyz, ix, iy, iz,
          ie, ia, octant, octant_in_block, noctant_per_block);
  }

for( ie=0; ie<dim_ne; ++ie )
{
  for( iu=0; iu<NU; ++iu )
    for( im=0; im<dim_nm; ++im )
      for( ia=0; ia<dim_na; ++ia )
      {
        result += ...;
        vs_local[<ind>] = result;
      }

#pragma omp atomic update
  vo_h[<ind>] += result;
}
```

Listing 1. `Sweeper_in_gricell()` parallelization using OpenMP 4.5

```
void Sweeper_sweep(...)
{

  /*--- Data transfer to the GPU ---*/
#pragma omp target enter data map(to: <vars>), map(alloc: facexy[...], facexz
      [...], faceyz[...])
#pragma omp target data map(to:facexy[,..])
{

#pragma omp target teams distribute parallel for collapse(3)
      for( octant=0; octant<NOCTANT; ++octant )
      for( iy=0; iy<dim_y; ++iy )
      for( ix=0; ix<dim_x; ++ix )
      for( ie=0; ie<dim_ne; ++ie )
      for( iu=0; iu<NU; ++iu )
      for( ia=0; ia<dim_na; ++ia )
      {

        /*--- ref_facexy inline ---*/
        /*--- Quantities_init_face routine ---*/
        faceyz[<ind>] = Quantities_init_face(ia, ie, iu, scalefactor_space,
              octant);
      }

 }

{

... // Repeats for facexy[], and facexz[]
#pragma omp target update from(faceyz[0:faceyz_size])
 }

#pragma omp target enter data map(alloc: vs_local[:vs_local_size]), map(to:<vars
      >)
 {

/*---Loop over octants---*/
for( octant=0; octant<NOCTANT; ++octant )
{
  ...

   /*--- Loop over wavefronts ---*/
   for (wavefront = 0; wavefront < num_wavefronts; wavefront++)
     {

/*--- Create an asynchronous queue for each octant ---*/

       /*---Loop over cells, in proper direction---*/
       if (dir_y==DIR_UP && dir_x==DIR_UP) {
         for( iy=0; iy<dim_y; ++iy )
           for( ix=0; ix<dim_x; ++ix )
             {
               /*--- In-gridcell computations ---*/
               Sweeper_in_gridcell(...);
             } /*---ix/iy---*/
       } else if (dir_y==DIR_UP && dir_x==DIR_DN) {
         for( iy=0; iy<dim_y; ++iy )
           for( ix=dim_x-1; ix>=0; --ix )
             {
               /*--- In-gridcell computations ---*/
               Sweeper_in_gridcell(...);
             } /*---ix/iy---*/
       } else if (dir_y==DIR_DN && dir_x==DIR_UP) {
         for( iy=dim_y-1; iy>=0; --iy )
           for( ix=0; ix<dim_x; ++ix )
             {
               /*--- In-gridcell computations ---*/
               Sweeper_in_gridcell(...);
             } /*---ix/iy---*/
       } else {

         for( iy=dim_y-1; iy>=0; --iy )
           for( ix=dim_x-1; ix>=0; --ix )
             {
               /*--- In-gridcell computations ---*/
               Sweeper_in_gridcell(...);
             }
       }

     }

     } /*--- wavefront ---*/

  } /*---octant---*/

 }   /*--- #pragma enter data ---*/


  /*--- Data transfer of results to the host ---*/
#pragma omp target exit data  map(from: vo_h[...]), map(delete: <vars>)


} /*---sweep---*/
```

Listing 2. `Sweeper_sweep()` function in OpenMP 4.5 version of Minisweep.

The majority of the OpenACC clauses in Minisweep are contained within the `Sweep_sweep()` and `Sweep_in_gridcell()` functions. In order to port Minisweep to the OpenMP 4.5 programming model, we employed the following transformations:

- `acc loop gang` clauses were replaced with `omp teams distribute` clauses.
- `acc loop vector` clauses were replaced with `omp parallel for` clauses.
- `acc loop seq` were removed as there is no equivalent clause in OpenMP 4.5.
- `acc parallel` regions were replaced with `omp target` regions.
- `acc wait` clause in `Sweep_sweep()` function was replaced with `omp taskwait`.
- `acc data copyout` clause was replaced with `omp data map(from:<var>)`.
- `acc data copyin` clause was replaced with `omp data map(to:<var>)`.
- `acc data create` clause was replaced with `omp data map(alloc:<var>)`.

2) Results: We used the following values in our single node, single GPU experiments for Minisweep. $n_e = 64, n_a = 32, n_u = 4, n_x = n_y = n_z = 32, NM = 16$. The same problem size was used in [10] which will allow us to compare the results to those from our OpenMP 4.5 prototype. Summit and Cori-GPU have NVIDIA V100 GPUs with 16GB of HBM2 memory and can support a 64^3 grid. Titan, on the other hand, has NVIDIA K20x GPUs with 6GB of DDR5 memory and can support only a 32^3 grid on a single GPU. For that reason, in this work we compare results from the 32^3 grid problem.

3) Lessons Learned: Due to the different interpretations of the individual specifications, in practice, we encountered that slight modifications of the source were required in order to successfully build and execute Minisweep on the different platforms using different compilers.

The GCC compiler with OpenACC support available on the target platforms was unable to compile the original OpenACC implementation of Minisweep and caused an internal compiler error (ICE). The following changes were required to successfully compile this version using GCC 7.1.1 on Summitdev, and GCC 8.1.1 on Summit and CoriGPU. First, we removed the `collapse(<number>)` clauses. Then, we also had to explicitly specify start and end segments in each `data` clause, e.g. `create(vs_local[0:size])` instead of `create(vs_local[size])`. The `gcc_openacc` branch [17] includes both changes. Although these changes allow us to build the code, the built-in verification step is not successful. We have included the performance results here for reference and continue to investigate the issue.

On Summit, we observed that the original OpenACC implementation caused unexpected behavior and will produce `Invalid device` errors. Because Summit has multiple GPUs per node and the job step launcher, `jsrun` assigns GPUs based on resource sets requested, using the `acc_device_default` clause can cause undefined results. Replacing that clause with `acc_device_nvidia` addressed the issue.

On Titan, while the Cray compiler CCE 8.6.4 is able to build the OpenACC implementation of Minisweep, we encountered runtime errors. We are still investigating the root cause of this issue. For that reason, only PGI was used on Titan for OpenACC experiments.

The OpenMP 4.5 version of Minisweep can be compiled successfully with the IBM XL C/C++ compiler on Summitdev and Summit. However, the performance is much slower than when using OpenACC or CUDA. The majority of the time is currently spent in data transfers between the host and the device. Alternative versions of the OpenMP 4.5 implementation are still being considered. Using CCE on Titan and CoriGPU the code fails to build. When using self-offloading on CoriKNL, we see that the code runs successfully but performance is significanlty lower than when using the GPUs.

B. GenASIS

Prior to this work, GENASIS has been ported to use GPU-acceleration for its hydrodynamics version via the use of OpenMP 4.5 device memory runtime library routines and `target` directive [18]. To achive useful speedup, the use of OpenMP to utilize GPU in GENASIS can be summarized by following two principles:

1) Explicit control of data allocation in GPU and their association with the host copy.
2) Explicit control of data movement between GPU and host.

GENASIS implements Fortran wrappers to several OpenMP 4.5 runtime library routines—because they are only provided in C—using the C interoperability capabilities of Fortran 2003 to implement the above principles. These wrappers provides lower-level GENASIS functionality—subroutines in the `Device` module—callable by higher-level classes / modules. They are:

- `AllocateDevice ( Value, D_Value )`: A call to this subroutine with the input argument of rank-1 or rank-2 array `Value` allocates a memory with the same array size on the GPU and sets the output variable `D_Value`—of `type ( c_ptr )`—to the allocated memory address on the GPU. Under the hood this subroutine callsthe OpenMP routine `omp_target_alloc()`.
- `DeallocateDevice ( D_Value )`: This is the counterpart of the subroutine `AllocateDevice ( )` to deallocate the memory region on the GPU by wrapping the OpenMP routine `omp_target_free()`

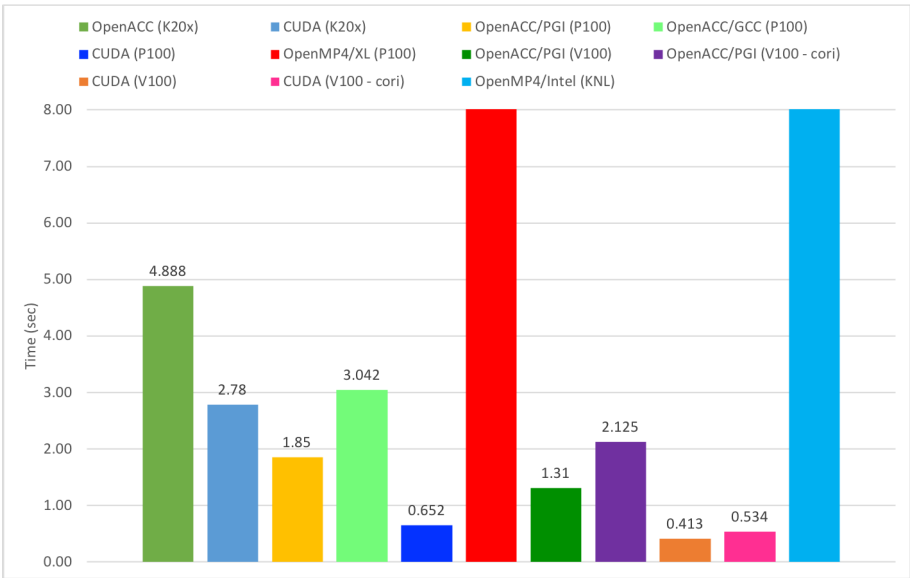

Figure 1. Minisweep timings on Summit (V100), Summitdev (P100), CoriGPU (V100), and Cori (KNL). The OpenMP 4.5 implementation performs considerably slower and its results go out of bounds of the figure.

- `AssociateHost ( D_Value, Value )`: A call to this subroutine associates the variable `Value` with device memory location in the variable `D_Value` such that for any reference to `Value` inside the offload directive, the device memory in `D_Value` is used. For our OpenMP implementation, this routine wraps `omp_target_associate_ptr()`.
- `DisassociateHost ( Value )` : This is the counterpart to `AssociateHost ( )` to remove the association of `Value` to any device memory location. This routine wraps OpenMP routine `omp_target_disassociate_ptr()`
- `UpdateDevice ( Value, D_Value )` : This routine copies data from the array `Value` to the device memory location in `D_Value` using OpenMP routine `omp_target_memcpy()`.
- `UpdateHost ( D_Value, Value )` : This routine copies the data from the device memory location in `D_Value` to the array `Value` using OpenMP routine `omp_target_memcpy()`.

Listing 3 shows an example of how some of the above subroutines are used within GENASIS In particular, within all of the kernels device memory location previously allocated or updated elsewhere are associated with host variable, such that any of the OpenMP offload directive avoids unintentional data transfer.

Porting the code to use OpenACC is therefore relatively straightforward, and can be summarized by the following steps:

1) Change / implement wrappers to call corresponding OpenACC library routines. These are, respectively:

```
subroutine ComputeDifference_X ( V, D_V, D_dV, dV )

  real ( KDR ), dimension ( -1:, -1:, -1: ), intent ( in ) :: V
  type ( c_ptr ), intent ( in ) :: D_V, D_dV
  real ( KDR ), dimension ( -1:, -1:, -1: ), intent ( out ) :: dV

  integer ( KDI ) :: i, j, k

  call AssociateHost ( D_V, V )
  call AssociateHost ( D_dV, dV )

  !$OMP  target teams distribute parallel do &
  !$OMP& collapse ( 3 ) schedule ( static, 1 )
  do k = 1, nZ
    do j = 1, nY
      do i = 0, nX + 2
        dV ( i, j, k )  =  V ( i, j, k )  -  V ( i - 1, j, k )
      end do
    end do
  end do
  !$OMP end target teams distribute parallel do

  call DisassociateHost ( dV )
  call DisassociateHost ( V )

end subroutine ComputeDifferences_X
```

Listing 3. A kernel for computing a nearest-neighbor difference.

- `acc_malloc()`
- `acc_free()`
- `acc_map_data()`
- `acc_unmap_data()`
- `acc_memcpy_to_device()`
- `acc_memcpy_from_device()`

2) Change OpenMP `target` directive to OpenACC `loop` directive. All of the kernels for this test problem have the OpenMP directive in the form of:
   - `!$OMP  target teams distribute &`
     `!$OMP& parallel do [collapse(n)] &`
     `!$OMP& schedule static ( 1 )`

   where the `collapse` clause is used for any nested

loop of depth n. The corresponding directive in OpenACC is simply:

- `!$ACC loop gang vector [collapse(n)].`

We implement a mechanism in GENASIS to switch between the OpenACC and OpenMP versions via preprocessor, macro, and Makefile such that compiling the different versions can simply be done by declaring a Makefile environment variable at build time. For example, the following commands build GENASIS's test problem `RiemannProblem` with OpenMP multithreading (default), OpenMP offload, and OpenACC, respectively:

- `make RiemannProblem`
- `make ENABLE_OMP_OFFLOAD=1 RiemannProblem`
- `make ENABLE_ACC_OFFLOAD=1 RiemannProblem`

For this experiment, we run an 3D extension of the classic `RiemannProblem` fluid dynamic problem on Summit, Titan, and Cori-GPU. In this case, we are mostly interested with performance on single GPU compared. The same code, however, has been demonstrated to run using up to 8000 GPU on Summit with good weak scaling using 1 MPI process per GPU. Figure 2 shows a visualization of this test problem when run with 125 GPUs with a total resolution of $1280\times1280\times1280$. We use $128\times128\times128$ cells per GPU for this experiment. We run the problem on Titan, Summit, and Cori-GPU for 50 time steps. We use the following compilers and versions:

- On Summit: XL-16.0.1 and GCC-8.1.1
- On Titan: CCE-8.7.7
- On Cori-GPU: CCE-8.7.7 and GCC-8.1.1

Figure 3 shows timings of various kernels required to solve the 3D `RiemannProblem` test problem with OpenMP for multi-threading on CPU, OpenMP offload with `target` directive, and OpenACC on Summi, Titan, and Cori-GPU. For the offload directives, as previously mentioned the bulk of the data transfers between the host and GPU is explicitly managed and measured as separate timing from the computational kernels. From this figure, we can glean that on Summit, speed-up from running on the GPU with offload directive is obtained with both XL and GCC compiler. On Titan however, both OpenMP and OpenACC offload with CCE yields slower execution. Interestingly, both seem to get the same execution timings, indicating a similar or even the same code is generated by the compiler despite using two different directives. On Cori-GPU, the results are mixed. While same speed-up is achieved with OpenACC using GCC compiler, the same thing cannot be said for the Cray CCE compiler with either directives.

Figure 4 shows more explicitly the speed-up or slow-down of the execution from code produced by multiple compilers with the different directives on each machine. On this Figure we plot the relative speed-up of the offloaded kernels to the multi-threaded version (using OpenMP) running on the CPUs of the respective machine. On Titan and Cori-GPU we use 8 CPU threads, while on Summit we use 7 CPU

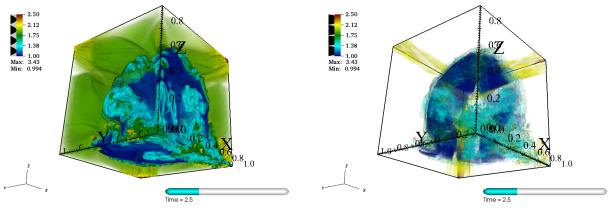

Figure 2. Volume plots of entropy of `RiemannProblem 3D` with $1280 \times 1280 \times 1280$ cells on 125 GPUs at time $t = 2.5$.

threads. As we can see on this Figure, on Summit the XL compiler with OpenMP offload produces an executable with speed-up of 10X or more for almost all kernels. With the GCC compiler using OpenACC on Summit, we also obtain a speed up for most kernels, albeit lesser, with the exception of the `Difference` and `Fluxes` kernels. On Titan with the CCE compiler however, all kernels show a slow down when OpenMP and OpenACC offload (not plotted on this Figure) directives are used. On Cori-GPU, the CCE compiler barely produces any speed-up for all the kernels using both directives. The GCC compiler on Cori-GPU yields some speed up for most kernels, but notably at a lesser factor in general compared to on Summit.

Since both Summit and Cori-GPU have the same GPU and the same version of GCC is used, we expect that the kernels to have similar timings when OpenACC is used on both machine. Figure 5 plots such timings. The results are somewhat surprising in that although many kernels have similar timings, several kernels show relatively large timing differences. These results merit further investigation outside the scope of this paper.

C. GPP

General Plasmon Pole (GPP) [19] is a mini-application from the BerkeleyGW [20] application suite. The GPP kernel computes the electron self-energy using General Plasmon Pole approximation. The kernel is comprised of 4-nested loops with a reduction in the innermost loop. Listing 4 shows the pseudo code of GPP.

```
for(num_bands){ //512
    for(ngpown){ //32768/20 = 1638
        for(ncouls){//32768
            for(iw){//3
                achtemp[iw] += ...;
            }
        }
    }
```

Listing 4. GPP pseudo code

The code uses a double-complex member as its primary data type and performs a series of tensor contraction like operations. The problem discussed in this paper has 512 electrons with 32,768 plane wave bases vectors and corresponds to a medium sized molecule. It is a common on-node problem size. This choice of size leads to the following characteristics for this kernel:

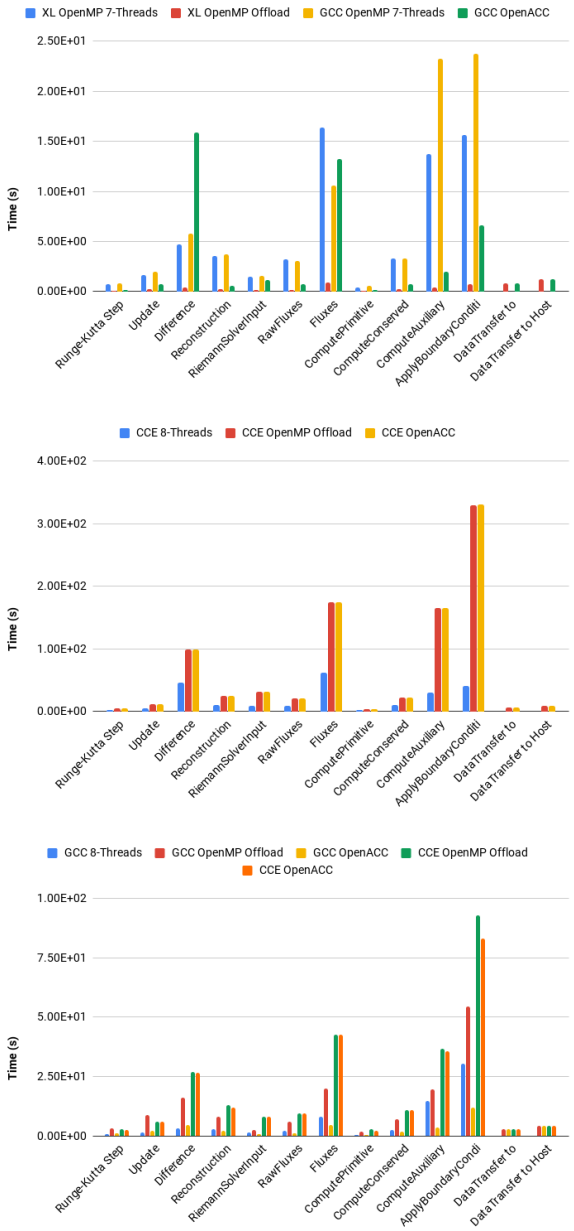

Figure 3. Timings of different kernels for `RiemannProblem 3D` on Summit (top), Titan (middle), and Cori-GPU (bottom).

- The overall memory footprint is approximately 2GB.
- The first and second loops are closely nested and can be collapsed. The resulting trip count will be around $\mathcal{O}(800K)$
- The third loop with the trip count of $\mathcal{O}(33K)$ can be vectorized for CPUs or parallelized with threads over GPUs
- The innermost loop has a small, fixed trip count of 3 and can be unrolled to exploit the SIMD/SIMT parallelization.

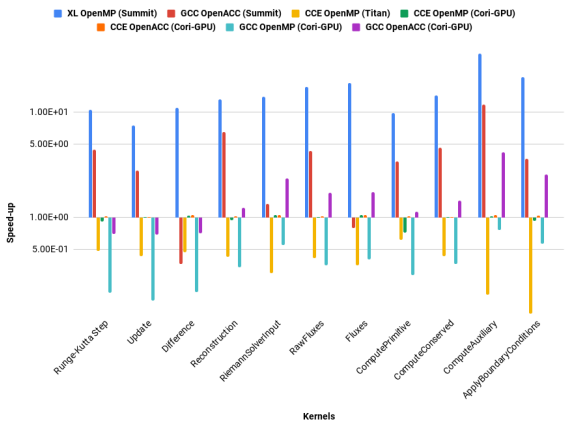

Figure 4. Speed-up (or slow-down) of various kernels on Summit, Titan, and Cori-GPU with multiple compilers and offload OpenMP or OpenACC directives relatives to 8-threads CPUs on the respective machines (7-threads on Summit). The y-axis is plotted in log scale. Value greater than 1.0 (with bars point up) represents speed-up, while value less than 1.0 (with bars point down) means slow down. Value equals to 1.0 represent no speed-up relative to the multi-threaded code.

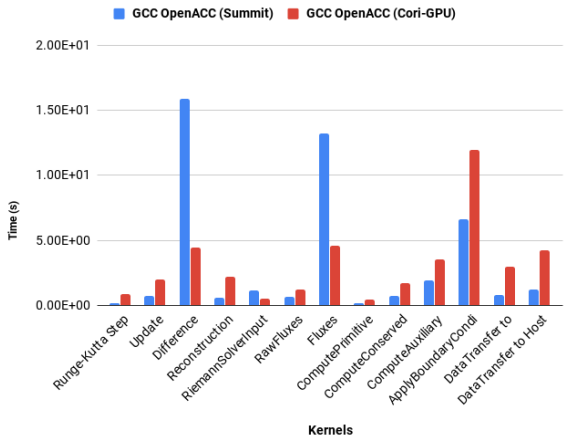

Figure 5. Timings of different kernels for `RiemannProblem 3D` with GCC-8.1.1 OpenACC directive on Summit (blue) and Cori-GPU (red).

In case of CPUs the most optimal parallelization technique was to distribute the outermost loop across the available threads. In case of OpenMP 4.5 and OpenACC for GPUs the most optimal implementations followed the following two steps:

1) Reverse the loop order of the first two loops.
2) Collapse the reversed loop order and distribute the resulting trip count across threadblocks and threads in a block.
3) In order to optimize the performance, we flatten the 3 complex numbers that form the final output into 6 scalars, representing 3 real and 3 imaginary parts and perform a *reduction* operation on them to maintain

correctness.

Listing 5 shows the most optimized OpenMP 4.5 implementation of the GPP kernel.

```
#pragma omp target teams distribute parallel for collapse(2) reduction(+:
      achtemp_re{0,1,2}, achtemp_im{0,1,2})
for(num_bands){ //512
    for(ngpown){ //32768/20 = 1638
        for(ncouls){//32768
            for(iw){//3
                //store local values
            }
            achtemp_re{0,1,2} += ...;
            achtemp_im{0,1,2} += ...;
        }
    }
```

Listing 5. GPP OpenMP 4.5 implementation

As shown in Listing 5, we flatten the three complex numbers *achtemp* into six scalars each of them representing three real and three imaginary parts of the complex numbers and perform a reduction on them. Although this makes the implementation less elegant, it has huge benefits to the performance. On CPUs, that support array reductions, we can perform a reduction on the real and imaginary array equivalents rather than the scalars. A detailed analysis of our methodology is presented in [21].

1) Results: GPP on Haswell takes 7.5 seconds and hence goes out of the bounds of the figure. The data transfers in the case of GPP and FF in both OpenMP 4.5 and OpenACC implementation were handled before the kernels began by using the data allocation and update statements provided by both the frameworks.

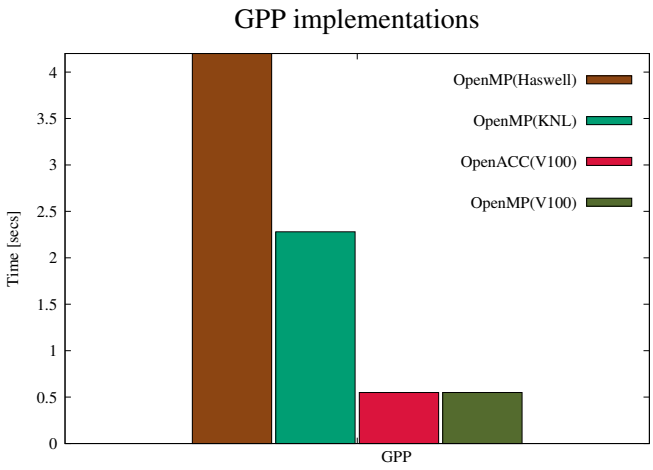

Figure 6. Results obtained from the different implementations of GPP.

2) Lessons Learned: We created two versions of the GPP application: one in more of a traditional C style and the other in a modern C++ style. The large memory objects in the C style application were arrays of a custom complex type, i.e. `CustomComplex<double> *aqsmtemp`, where `aqsmtemp` was allocated using `new`. The data for these large memory objects could easily be copied to the device using `map(aqsmtemp[0:N])`. This method worked well across all compilers. On the other hand, we encountered many compiler errors when using a modern C++ style where we replaced a C array of Complex type with a C++ array class of Complex type, i.e. `Array2D<CustomComplex<double>> aqsmtemp`. The only compiler which consistently supported the modern C++ style was the upstream LLVM/Clang compiler. The errors we encountered included:

1) The Cray compiler was unable to map C++ references to the device.
2) The IBM compiler hung in the OpenMP device kernel.
3) The PGI compiler did not support a function call in the OpenMP map construct, e.g. `aqsmtemp.dptr[0:aqsmtemp.getSpan()]`, where `getSpan()` was an inline function returning the number of elements in the C++ array object.
4) The GNU compiler failed at link time.

This highlights the challenge of writing modular and abstract code in combination with the OpenMP target offload model. The lack of support for C++ reference types in the Cray compiler forces the programmer to declare variables in the same code unit as the operations on that variable, i.e. preventing the passing of data references to child functions. This hurts modularity. This work also demonstrated that the only robust way to use C++ with OpenMP target offload across compilers was to only use C arrays containing plain-old-data (POD) types on the device. Once again this highlights a tension between good software engineering practices in C++ and the restricted subset of C++ which can successfully be used with multiple OpenMP compilers.

Although not shown in the results, we ran many performance comparisons using the C style version of GPP across compilers. We found that the `simd` construct needed to be added when using the Cray compiler to obtain reasonable performance. Here, the Cray compiler auto-parallelizer failed to map an OpenMP parallel for loop to the threads on the GPU. Performance improved by 80x when adding the `simd` construct.

D. FF

FF kernel represents the Full-Frequency Self-Energy Summations in BerkeleyGW. It has 3 kernels each with its own loop structure. The loop structures of all three kernels in FF are shown below:

```
#pragma omp target teams distribute parallel for collapse(2) reduction(+:
      achsDtemp_re, achsDtemp_im)
for(num_bands){ //15023
    for(ngpown){ //66
        for(ncouls){//22401
            achsDtemp += ...;
            }
        }
    }
    achsDtemp = CustomComplex<double> (achsDtemp_re, achsDtemp_im);
```

Listing 6. achsDtemp Kernel

```
#pragma omp target teams distribute parallel for collapse(3)
for(num_bands){ //15023
    for(ngpown){ //66
        for(iw){//10
            for(ncouls){//22401
            #pragma omp atomic
                        asxDtemp[iw] += ...;
                }
            }
        }
    }
```

Listing 7. asxDtemp Kernel

```
#pragma omp target teams distribute parallel for collapse(2)
for(num_bands){ //15023
    for(iw){ //66
        for(ngpown){//10
            for(ncouls){//22401
            #pragma omp atomic
                        achDtemp_cor[iw] += ...;
                }
            }
        }
    }
```

Listing 8. achDtemp_cor Kernel

As shown in the kernel listings, all three kernels have a different loop structure. While in the *achsDtemp* kernel, the reduction is on a single complex variable, in *asxDtemp* and *achDtemp_cor* the reduction is on an array of 10 elements. In those cases, we used atomics to maintain correctness. The OpenMP 4.5 parallelization technique for each of the kernels is shown in the Listings **??**. The optimized OpenACC implementation follows a similar parallelization technique and we are able to replace the OpenMP 4.5 directives with their equivalent OpenACC directives.

1) Results: In case of FF and GPP kernels, the OpenMP 3.0 version is compiled with the Intel compilers, the OpenMP 4.5 with the Clang compiler and OpenACC with the PGI compiler.

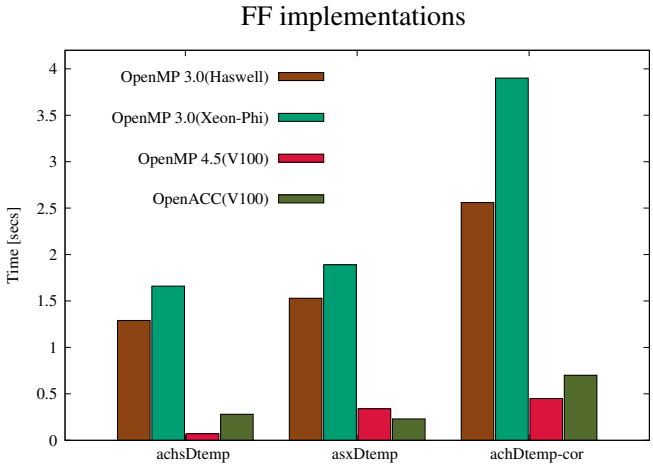

Figure 7. Results obtained from different implementations of FF.

2) Lessons Learned: The lessons learned from porting and running the FF kernel are summarized in Section II-C2. Since both kernels, FF and GPP, perform a set of complex number calculations we can learn from one of them and apply the findings to the other kernel. The major difference between the kernels is the use of *atomics* in the FF kernel. Another major observation in the case of FF is that the OpenMP 4.5 offloading on GPUs consistently performs better than OpenACC on all 3 kernels. We are still in the process of understanding this behavior.

III. Conclusion

In this work, we provide an overview of our experiences porting four mini-applications of importance to the OLCF and NERSC. The mini-applications chosen—Minisweep, GENASIS GPP, and FF—are representative of real HPC applications and use algorithmic patterns common to several codes.

Some of the codes used in this study had already been ported to one of the two programming models of interest. This work contributes an OpenMP 4.5 port of Minisweep, and an OpenACC port of GENASIS.

For GENASIS, although porting from the OpenMP 4.5 to OpenACC is relatively straightforward, the same performance for the two versions of the code is not obtained. This may be the due to different compiler's maturity in implementing directives or to the fact that more tuning specific to the particular programmimg model is needed to achieve good performance.

Similarly for Minisweep, which started with an OpenACC implementation, porting has proved to require efforts to update the code and to workaround several compiler-related issues. Although some features are part of the language standard or directive specification, in reality, compiler's support varies and one must adapt the code to be able to use a certain compiler with a particular programming model. Yet again this speaks to the different maturity of compilers in implementing directives, often making porting efforts greater than necessary. Similar lessons are present in the efforts to port both the GPP and FF mini-applications.

We have documented these lesson learned in our porting efforts and believe they will be useful for the community and compiler vendors implementing these directives. The codes made available from this work may also benefit the HPC community at large.

Acknowledgment

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] TOP500, "November 2018 list," 2019. [Online]. Available: https://www.top500.org/list/2018/11/

[2] V. G. V. Larrea, W. Joubert, M. G. Lopez, and O. Hernández, "Early experiences writing performance portable openmp 4 codes," 2016.

[3] S. Boehm, S. S. Pophale, V. G. Melesse Vergara, and O. R. Hernandez, "Evaluating performance portability of accelerator programming models using spec accel 1.2 benchmarks," Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), Tech. Rep., 2018.

[4] G. Juckeland, O. Hernandez, A. C. Jacob, D. Neilson, V. G. V. Larrea, S. Wienke, A. Bobyr, W. C. Brantley, S. Chandrasekaran, M. Colgrove, A. Grund, R. Henschel, W. Joubert, M. S. Müller, D. Raddatz, P. Shelepugin, B. Whitney, B. Wang, and K. Kumaran, "From describing to prescribing parallelism: Translating the spec accel openacc suite to openmp target directives," in High Performance Computing, M. Taufer, B. Mohr, and J. M. Kunkel, Eds. Cham: Springer International Publishing, 2016, pp. 470–488.

[5] R. Searles, S. Chandrasekaran, W. Joubert, and O. R. Hernandez, "MPI + openacc: Accelerating radiation transport mini-application, minisweep, on heterogeneous systems," Computer Physics Communications, vol. 236, pp. 176–187, 2019. [Online]. Available: https://doi.org/10.1016/j.cpc.2018.10.007

[6] O. Hernandez, W. Ding, B. Chapman, C. Kartsaklis, R. Sankaran, and R. Graham, "Experiences with high-level programming directives for porting applications to gpus," in Facing the Multicore-Challenge II. Springer, 2012, pp. 96–107.

[7] J. M. Levesque, R. Sankaran, and R. Grout, "Hybridizing s3d into an exascale application using openacc: an approach for moving to multi-petaflops and beyond," in Proceedings of the International conference on high performance computing, networking, storage and analysis. IEEE Computer Society Press, 2012, p. 15.

[8] O. B. Messer, E. D'Azevedo, J. Hill, W. Joubert, M. Berrill, and C. Zimmer, "Miniapps derived from production hpc applications using multiple programing models," The International Journal of High Performance Computing Applications, vol. 32, no. 4, pp. 582–593, 2018. [Online]. Available: https://doi.org/10.1177/1094342016668241

[9] C. G. Baker, G. G. Davidson, T. M. Evans, S. P. Hamilton, J. J. Jarrell, and W. Joubert, "High performance radiation transport simulations: Preparing for TITAN," in Proceedings of Supercomputing Conference SC12, 2012.

[10] R. Searles, S. Chandrasekaran, W. Joubert, and O. Hernandez, "Mpi+ openacc: Accelerating radiation transport mini-application, minisweep, on heterogeneous systems," Computer Physics Communications, vol. 236, pp. 176–187, 2019.

[11] C. Y. Cardall, R. D. Budiardja, E. Endeve, and A. Mezzacappa, "GENASIS: GENERAL ASTROPHYSICAL SIMULATION SYSTEM. i. REFINABLE MESH AND NONRELATIVISTIC HYDRODYNAMICS," The Astrophysical Journal Supplement Series, vol. 210, no. 2, p. 17, jan 2014. [Online]. Available: https://doi.org/10.1088%2F0067-0049%2F210%2F2%2F17

[12] C. Y. Cardall and R. D. Budiardja, "Genasis basics: Object-oriented utilitarian functionality for large-scale physics simulations," Computer Physics Communications, vol. 196, pp. 506 – 534, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465515002453

[13] R. D. Budiardja and C. Y. Cardall, "Targeting gpus with openmp directives on summit: A simple and effective fortran experience," arXiv preprint arXiv:1812.07977, 2018.

[14] J. Deslippe, G. Samsonidze, D. A. Strubbe, M. Jain, M. L. Cohen, and S. G. Louie, "Berkeleygw: A massively parallel computer package for the calculation of the quasiparticle and optical properties of materials and nanostructures," Computer Physics Communications, vol. 183, no. 6, pp. 1269 – 1289, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465511003912

[15] "TOP500 Supercomputer Sites," https://www.top500.org.

[16] "Ornl-cees," https://github.com/ORNL-CEES/Profugus, 2017.

[17] "Olcf minisweep," https://github.com/olcf/minisweep, 2019.

[18] R. D. Budiardja and C. Y. Cardall, "Targeting GPUs with OpenMP Directives on Summit: A Simple and Effective Fortran Experience," arXiv e-prints, p. arXiv:1812.07977, Dec 2018.

[19] J. Soininen, J. Rehr, and E. L. Shirley, "Electron self-energy calculation using a general multi-pole approximation," Journal of Physics: Condensed Matter, vol. 15, no. 17, p. 2573, 2003.

[20] J. Deslippe, G. Samsonidze, D. A. Strubbe, M. Jain, M. L. Cohen, and S. G. Louie, "Berkeleygw: A massively parallel computer package for the calculation of the quasiparticle and optical properties of materials and nanostructures," Computer Physics Communications, vol. 183, no. 6, pp. 1269–1289, 2012.

[21] R. Gayatri, C. Yang, T. Kurth, and J. Deslippe, "A case study for performance portability using openmp 4.5," in International Workshop on Accelerator Programming Using Directives. Springer, 2018, pp. 75–95.